Summer Student Project Report: CMS Level-1 Global Trigger Algorithm Rate Monitoring

Petar Sušac
Supervisors: Elias Leutgeb, Dinyar Rabady

01 September 2023

1 Introduction

The Phase-2 upgrade of the Compact Muon Solenoid (CMS) experiment Level-1 trigger (L1T) is set to enhance physics selectivity of the trigger system at the hardware level to fully exploit the capabilities of the planned High Luminosity Large Hadron Collider (HL-LHC). The HL-LHC will achieve greater luminosity, while the upgraded CMS detector will enable extended coverage and increased granularity [5].

A functional diagram of the Phase-2 Level-1 trigger is shown in Figure 1. It consists of several subsystems: the Calorimeter Trigger, Muon Trigger, Track Trigger, Correlator Trigger and the Global Trigger. The final trigger decision is made by the Phase-2 Global Trigger (P2GT) based on more than 20 different trigger object collections from upstream systems. The P2GT will be able to evaluate about 1000 cut-based and machine learning algorithms.

The functional structure of the P2GT is shown in Figure 2. Its inputs are time-multiplexed collections of objects from upstream systems. These collections are demultiplexed and trigger algorithms are applied. In its final implementation, the P2GT will consist of 12 algorithm boards.

Figure 3 shows a high-level structure of one of the P2GT algorithm boards. The result of every algorithm calculation can be either 0 or 1, where 1 is a positive result. These bits are stored in a signal called `algo_bits`, which is then output on an optical link to the Final OR board.

The goal of this project was to implement a rate counter module. Its purpose is to count how many times each algorithm produced a positive result ("fired") during one orbit. This module will be used to monitor the performance of the algorithms during the algorithm board test procedures.

In addition, a UART hardware testbench was developed to enable testing each individual algorithm, isolated from other system components, on a scaled-down hardware setup.

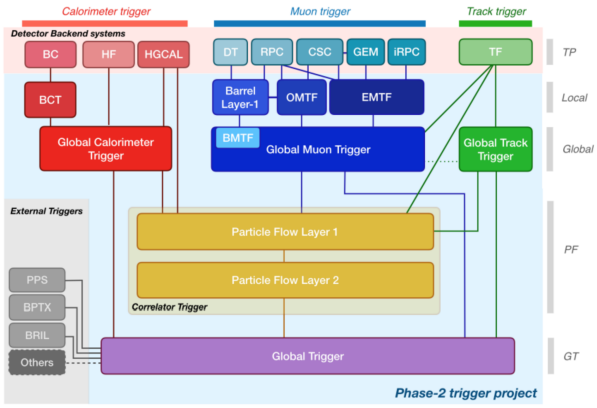

Figure 1: Functional diagram of the Level-1 trigger

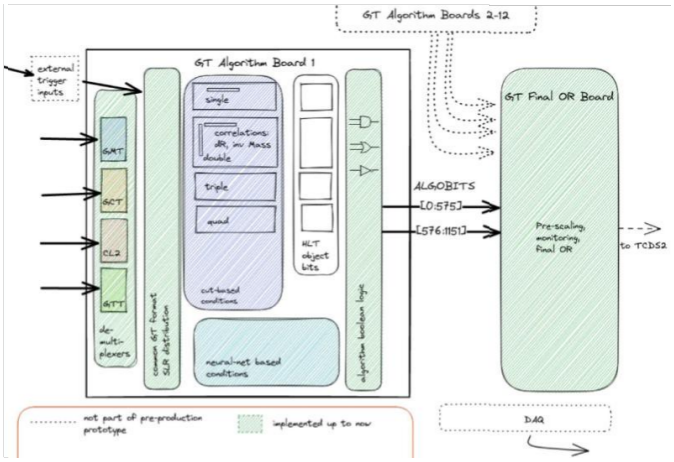

Figure 2: Functional sketch of the Phase-2 Global Trigger

2 Rate counter

The rate counter module keeps track of how often each trigger algorithm fires. The results of algorithm calculations are stored inside the `p2gt_algos` module in a signal named `algo_bits`. The width of this signal corresponds to the number of implemented algorithms. The validity of `algo_bits` is indicated by the signal `algo_bits_valid`. Trigger algorithms operate with a clock frequency of 480 MHz, which means that there are 12 algorithm clock cycles for each event arriving at a rate of 40 MHz, i.e., `algo_bits` will change every 12 algorithm clock cycles.

The rate counter contains a counter for each algorithm. The counter is incremented once per event if the algorithm produced a positive result. When

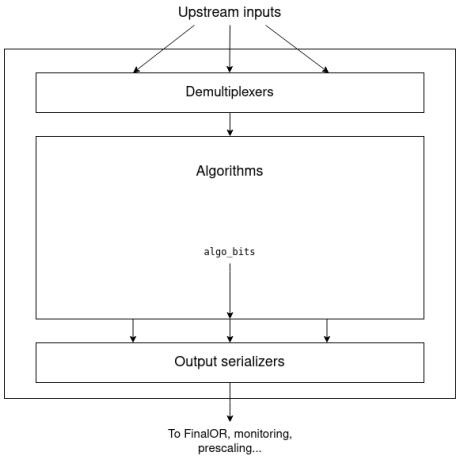

Figure 3: Functional sketch of an algorithm board

the `algo_bits_valid` signal goes low at the end of orbit, indicating there are no more events to process, the rate counters values are written to the output link and the counters are reset.

Since the `algo_bits` signal is driven by a 480 MHz algorithm clock but only changes every 12 clock cycles, a multicycle path constraint is used to relax the timing requirements enforced by the synthesis tool.

Figures 4 and 5 show an example of using multicycle constraints to modify the setup and hold relationships between a fast launch clock and a slow capture clock. In this example, the launch clock is 3 times faster than the capture clock. Assuming the signal will remain constant for 3 cycles of the launch clock, the edge used for setup check can be moved 3 edges forward relative to the launch clock. The hold edge is moved automatically with the setup edge, so it will remain one launch cycle ahead. Therefore, an additional constraint should be defined to move it backward so that the setup and hold checks are performed on the same capture clock edge.

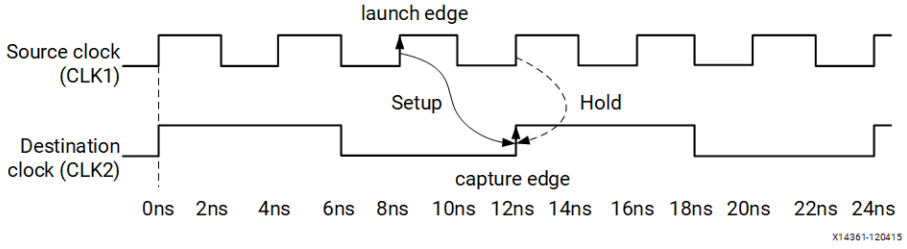

Figure 4: Setup and hold relationships before applying multicycle constraints [6]

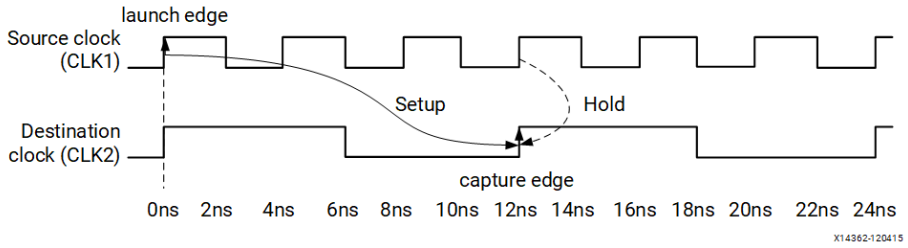

Figure 5: Setup and hold relationships after applying multicycle constraints [6]

In case of the implemented rate counter, the algorithm clock is 12 times faster than the rate counter clock (LHC clock) so the setup check is moved forward by 12 clock cycles and the hold check is moved backward by 11 clock cycles. Similarly, when crossing from a slow to a fast clock domain (rate counter outputs captured by the algorithm clock) multicycle constraints are set so that the setup check is moved forward 12 clock cycles and the hold check is moved back 11 clock cycles (aligned with launch clock edge). This allows the signal enough time to propagate between two capture edges without violating timing requirements. The constraints are set with the following code snippet.

```
set_multicycle_path 12 -setup -start -from [get_clocks -filter {
    NAME =~ clks_aux_u_0* }] -to [get_clocks -filter {NAME =~
    clks_aux_u_2* }]
set_multicycle_path 11 -hold -from [get_clocks -filter {NAME =~
    clks_aux_u_0* }] -to [get_clocks -filter {NAME =~ clks_aux_u_2*
     }]

set_multicycle_path 12 -setup -from [get_clocks -filter {NAME =~
    clks_aux_u_2* }] -to [get_clocks -filter {NAME =~ clks_aux_u_0*
     }]
set_multicycle_path 11 -hold -end -from [get_clocks -filter {NAME
    =~ clks_aux_u_2* }] -to [get_clocks -filter {NAME =~
    clks_aux_u_0* }]
```

Performance of the rate counter module was tested and validated in both simulation and hardware.

3 Monitoring module

In addition to having the rate counter values available on the output link, it would be useful to be able to read them using the IPbus interface [2]. A monitoring module was already implemented for a similar purpose on the Final OR board, providing algorithm rate counting, prescaler settings and other monitoring features along with IPbus communication. This module was successfully ported to the algorithm board using the rate counter module discussed in the previous section as a wrapper.

Figure 6 shows the final structure of the algorithm board. These additions to

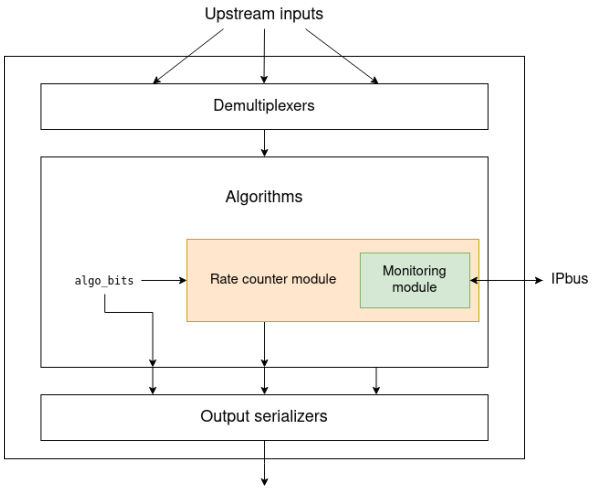

Figure 6: Block diagram of the algorithm board with rate counter module

the firmware allow the algorithms to be monitored directly from the algorithm boards, using IPbus or by monitoring the output links, without the need for a separate monitoring board.

Due to limited space on the FPGA chip, the number of monitored algorithms is limited. The module was successfully tested with 32 and 64 algorithms, and more algorithms can possibly be added in case they are needed. Figure 7 shows the FPGA floorplan for the KU15P demonstrator board with the implemented monitoring modules with 32 and 64 algorithms, respectively. The monitoring module itself is depicted in pink, while the rest of the rate counter module, which outputs rate counters on the link and is independent of the monitoring module, is depicted in red. It can be concluded that doubling the number of algorithms has approximately doubled the size of the monitoring module, while the rest of the rate counter module remained the same size.

Algorithm bits are routed through the rate counter module to the monitoring module. Since the monitoring module does not rely on a valid signal but instead expects all the algorithm bits to be pulled to zero when they are not valid, the algorithm bits are AND-ed together with the algorithm bits valid signal before being passed to the monitoring module to ensure this condition is satisfied.

The IPbus interface was added to the algorithm board payload to support the monitoring module. Counter registers can be read via IPbus as a memory block using software libraries. The values of these counters are updated once per luminosity section. In addition, prescalers and trigger masks can be set. Correct behavior of the module was verified in hardware.

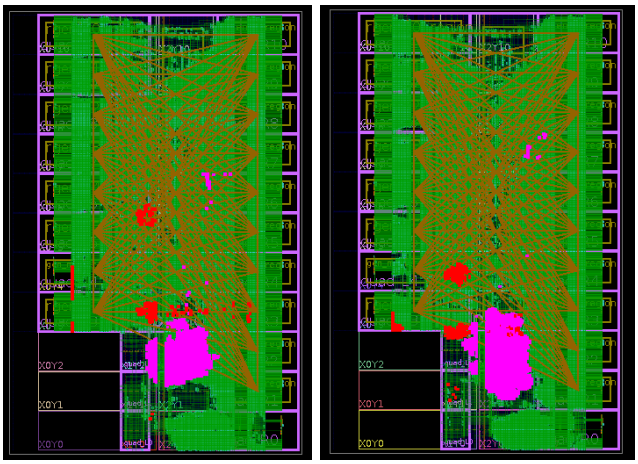

Figure 7: Floorplan of KU15P FPGA. Highlighted are monitoring modules with 32 algorithms (left) and 64 algorithms (right).

4 Remote monitoring

The P2GT algorithm firmware runs on the Serenity board [4]. The Serenity is a carrier card with two daughter FPGA card ports and a CPU running CentOS 7. This CPU can be remotely accessed and used to program the FPGA. As part of this project, a web server was set up on the on-board CPU to allow remote monitoring and communication with the FPGA hardware.

4.1 uHAL

uHAL (Micro Hardware Access Layer) is a library that provides a C++ and Python API for IPbus transactions. To communicate via IPbus an address table needs to be provided in the form of an XML file that specifies the addresses of registers and memory blocks that are available for IPbus read/write operations.

An example of using uHAL in Python can be seen below. This function reads the values of rate counters from a monitoring module register. `get_hw_device()` is a wrapper function that calls uHAL methods to set the connection file, initialize a connection manager and connect to the hardware. The `hw.getNode()` method retrieves a register by finding the register's name in the address table, and `readBlock(size)` reads the value of the register given its size in 32-bit words. Since the values of rate counters update once per lumi section, they are divided by the number of orbits in one lumi section (which is 2^{18}) to get the values of the rate counters per one orbit. This kind of calculation is possible because, in the test setup, the same input data is sent each orbit.

```
def read_counters(size=32, after_prsc=True):
    hw = get_hw_device()
    if after_prsc:
```

```
  rate_cntrs = hw.getNode(RATE_CNT_AFTER_PRSC_REG).readBlock(
size)
else:
  rate_cntrs = hw.getNode(RATE_CNT_BEFORE_PRSC_REG).readBlock(
size)
hw.dispatch()
rate_cntrs_norm = [int(x / 2**lumi_bit) for x in rate_cntrs]
return rate_cntrs_norm
```

4.2 Web interface

A web server was implemented to allow remote access while running the monitoring application on the Serenity on-board CPU. The server is implemented using the Django framework for Python. HTTP requests from the user trigger access to the hardware registers via IPbus. The server serves two web pages: one that plots the values of the rate counters before and after prescalers, and another that allows the user control over prescalers. For security reasons, access to the interface is allowed only to machines with IP addresses within the same subnet as the server.

The user is able to set the values of prescale registers to regulate how often each algorithm fires through a web form. The current values of prescale registers, read from hardware, are also displayed. The values of rate counters before and after prescalers are retrieved from hardware registers and plotted using the Chart.js JavaScript library, as shown in Figure 8.

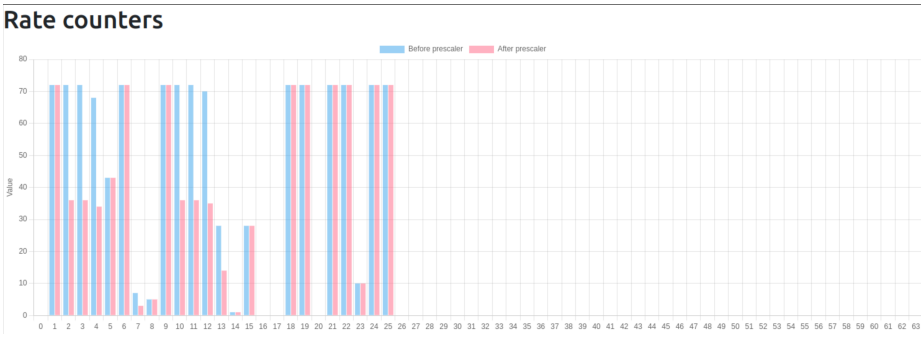

Figure 8: Plot of the rate counter values before and after prescalers in the web interface

4.3 Containerization and continuous integration

The web server itself runs in a Docker container to provide isolation from other programs running on the Serenity board. The container image is based on the `serenity-sw-dev-centos7` image available in the CERN GitLab registry [3]. A GitLab Continuous Integration (CI) pipeline has been set up to build a Docker image containing the web app and all its dependencies and upload it to the

registry. The CI script then uses SSH to log into the Serenity board, pull the image and run the container. This way, any changes to the web server code are automatically deployed on the Serenity.

5 UART test module

In order to enable testing the performance of individual algorithms on a hardware platform without the need to build the entire system, a test firmware module was implemented to be used with the Digilent Cmod A7-35T development board featuring an Artix-7 FPGA module. The Cmod A7-35T also features a USB-UART bridge, which was used to establish communication between the FPGA module and a PC.

Figure 9: Cmod A7-35T development platform [1]

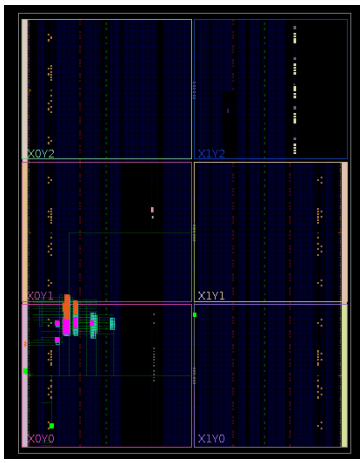

Figure 10: Implemented design of UART test module

The UART receiver and transmitter modules were implemented in FPGA firmware using VHDL. A state machine provides functionality needed for testing the trigger algorithms. Input data can be sent from the PC via USB (using a serial terminal emulator such as *minicom* or *CoolTerm*) in the form of 8-bit words. The FPGA module receives these words via a USB-UART bridge and

stores them in registers. Upon receiving a predefined number of inputs, the algorithm calculation is performed and the result (0 or 1) is sent back to the PC via UART as a single byte.

Figure 10 shows the implemented FPGA design. Logic blocks corresponding to the UART RX component are highlighted in pink, while those corresponding to the UART TX component are highlighted in orange. The rest of the configurable logic blocks are available for the implementation of trigger algorithms.

References

[1] *Cmod A7-35T: Breadboardable Artix-7 FPGA Module*. Digilent. URL: https://digilent.com/shop/cmod-a7-35t-breadboardable-artix-7-fpga-module/.

[2] C. Ghabrous Larrea et al. "IPbus: a flexible Ethernet-based control system for xTCA hardware". In: *JINST* 10.02 (2015), p. C02019. DOI: 10.1088/1748-0221/10/02/C02019.

[3] *Phase-2 software Docker images*. URL: https://gitlab.cern.ch/p2-xware/software/docker-images.

[4] A. Rose et al. "Serenity: An ATCA prototyping platform for CMS Phase-2". In: *Topical Workshop on Electronics for Particle Physics (TWEPP2018)*. CERN. Antwerp, Belgium, Sept. 2018, pp. 1–4. URL: https://serenity.web.cern.ch/TWEPP18_Serenity_Rose.pdf.

[5] *The Phase-2 Upgrade of the CMS Level-1 Trigger*. Tech. rep. Final version. Geneva: CERN, 2020. URL: https://cds.cern.ch/record/2714892.

[6] *Vivado Design Suite User Guide: Using Constraints (UG903)*. Version 2021.2. Xilinx. URL: https://docs.xilinx.com/r/2021.2-English/ug903-vivado-using-constraints/Multicycle-Paths.